Advanced UNIX Environment

B. Estrade <estrabd@lsu.edu>
Opening Thoughts

• To truly master computers, one needs to be proficient at, and understand, the command line – there's no way around this.

• The UNIX learning curve is steep, but one day you 'just get it'. So, keep trying until that happens.

• UNIX is fun because it is powerful and enabling! :)... and as one progresses, an appreciation will develop for its elegance and power.
Operating System

• What is an Operating System?
  – A program that *controls the execution of other (application) programs* – though lines are blurred
  – An interface between applications and hardware

• Primary functionality
  – Exploits the hardware resources of one or more processors
  – Provides a set of services to system users
  – Manages secondary memory and I/O devices

• Objectives
  – *Convenience*: Makes the computer more convenient to use
  – *Efficiency*: Allows computer system resources to be used in an efficient manner
  – *Reliability*: through protection between jobs
  – *Ability to evolve*: Permit effective development, testing, and introduction of new system functions without interfering with service
Operating System Structure

• System Components
  – Process Management
  – Main-Memory Management
  – File Management
  – I/O System Management
  – Secondary Storage Management
  – Networking
  – Protection & Security Systems
Brief History of UNIX

- Initially developed at Bell Labs in late 1960s by a group including Ken Thompson, Dennis Ritchie and Douglas McIlroy
- Originally named *Unics* in contrast to *Multics*, a novel experimental OS at the time
- The first deployment platform was PDP-7 in 1970
- Rewritten in C in 1973 to enable portability to other machines (most notably PDP-11) – an unusual strategy as most OS’s were written in assembly language
- Version 6 (version numbers were determined by editions of system manuals), released in 1976, was the first widely available version outside Bell Labs
- Version 7 (1978) is the ancestor of most modern UNIX systems
- The most important non-AT&T implementation is UNIX BSD, developed at the UC at Berkeley and to run on PDP and VAX
- By 1982 Bell Labs combined various UNIX variants into a single system, marketed as UNIX System III, which later evolved into System V
Traditional UNIX Organization

- Hardware is surrounded by the operating system software
- Operating system is called the system kernel
- Comes with a number of user services and interfaces
  - Shell
  - Components of the C compiler
Command – Interpreter System

- The command interpreter is the interface between the user and the operating system a.k.a., the shell.
- Some operating systems include a shell in the kernel while others such as MSDOS* and UNIX treat the shell as a special program.
- Commands are given to the operating system by a control statement issued by a user (e.g.: `ls`, `rm`, `mv`) interactively.
- These control statements are interpreted by the shell, whose main function is to get the next command statement and execute it.
- Some operating systems offer graphical interfaces (GUIs) to perform the same operations.
- Some GUIs are built into the kernel (Windows), while others are not (X11 on UNIX).
What is The Shell?

- So, the shell is a high level interface to the operating system for users.
- *This is “the prompt” that you get when you login.*
- Different shells are preferred different users, but they all provide the same interactive and *batch* access to the underlying OS.
How Does It Work?

1) The user issues shell command at the shell prompt.
2) The shell sends a lower level command to the OS.
3) The OS executes the command and returns any results back to the user via the shell.
Features of The Shell

• All modern, *useful* shells provide typical language constructs (comparison, flow control, etc).
• A set of shell commands and constructs can be saved into a text file and be run as a program; these are called *shell scripts*.
• Shells can track global variables that are referred to as their “*environment***”.
What is The Environment?

- The shell's environment is that umbrella of persistent knowledge under which the shell operates.
- This information is stored as "environmental variables," which are global variables that may be used at any point by shell – either internally or when explicitly required by the user.
- Some variables are set when one first logs-in;
- Other variables can be customized by the user using a specific set of files contained in their HOME directory.
What is The Environment?

Global Environment

- **PATH**=/usr/bin:/etc:/usr/sbin:/usr/local/bin
- **LOGIN**=username
- **PWD**=/home/username
- **HOME**=/home/username
- **USER**=username

Local environments may override the initial global environment.
Viewing The Environment

• View the entire environment

  `env | more` # use of pager, more

• View a specific variable (*more than 1 way*)

  `printenv VARNAME` # option 1, no $

  `echo $VARNAME` # option 2, $
Notable Environmental Variables

- **HOME**
  - Your home directory (e.g., `% cd $HOME`)

- **PATH**
  - A colon delimited list of paths that are searched for executables. It's used by shell and tools like `which`.

- **LD_LIBRARY_PATH** (*LIBPATH on IBM AIX*)
  - A colon delimited list of paths to look for libraries (replaces `-l` when linking C or Fortran programs);

- **LD_INCLUDE_PATH**
  - A colon delimited list of paths to look for include files (replaces `-I` when linking C or Fortran programs).
Not All Shells Are Created Equally

- Be aware that different shells behave differently and have different commands for similar functionality.
- The 2 common families
  - **Bourne** Shell:
    - `sh, bash, ksh, zsh`
    - *good as a login shell and as a basis for a shell script program*
  - **C-Shell**:
    - `csh, tcsh`
    - *good only as a login shell; avoid using it to program shell scripts*

http://www.faqs.org/faqs/unix-faq/shell/csh-whynot
http://www.grymoire.com/Unix/CshTop10.txt
Manipulating The Environment

• One may create or modify a global environmental variable;
  – Use “export” command in bash;
  – Use “setenv” command in csh;

• Examples,

  % export VARNAME='value' # bourne shell
  % setenv VARNAME 'value' # c-shell
PATH Variables

• Use **PATH** to store multiple paths used for automatically searching for executables, etc.

• **NB**: Extra care must be taken when modifying these variables because in most cases you should simply add to the existing value.

• Examples,

  ```
  % export PATH="/my/new/path:${PATH}" # bourne shell
  % setenv PATH "/my/new/path:${PATH}" # c-shell
  ```
Creating a Custom Environment

• When one logs in interactively, the system default environment is set using the following files:
  – /etc/profile    # bourne shell
  – /etc/csh.cshrc  # csh
• These files automatically set up the default path and vital user system variables.
Customizing the Environment

• bash users may create the following files in their home directory:
  – `~/.bash_profile` # 1. run first after `/etc/profile` if exists**
  – `~/.bashrc` # 2. run after `~/.bash_profile` if exists
  – `~/.profile` # 3. run after `~/.bashrc` if exists

• bash users may also create a file that is executed when one logs out:
  – `~/.bash_logout`

** ~/.bash_profile must not produce any standard out, since it will break tools such as rsync.
The File System

- UNIX treats the file system as a unified tree with a single root, from which there is a unique absolute path to every directory and file.
- This is as opposed to Windows, which treats each drive as the root of a tree.
- Many disks may be “mounted” into this single unified view.
- The file system is made up of directories and files, as expected;
- The OS tracks a lot of information behind the scenes, including:
  - mount points for physical and logical drives
  - *inodes*, which are unique identifiers for each directory and file
Typical *Top Level* Directory Layout

- `/`
  the root partition (there is only the one root)
- `/dev`
  device files, e.g., `/dev/zero`, `/dev/urandom`, `/dev/ads01a`, etc
- `/usr`
  main “user” directory, e.g., `/usr/bin`, `/usr/lib`, `/usr/home`, etc
- `/var`
  tracking the system, e.g., `/var/mail/spool`, `/var/log/syslog`, etc
- `/tmp`
  for temporary data
- `/etc`
  system settings, e.g., `/etc/profile`, `/etc/cshrc.csh`, `/etc/hosts`, etc
- `/mnt`
  provides place to “mount” peripherials, e.g., `/mnt/thumbdrive`
Directory Path = Path in Tree

/ /dev /usr /var /tmp /etc /mnt

/bin /home /lib /local /share

/user1 /user2 /user3 /user4 /user5

/usr/home/user5
UNIX File Types

- **Regular**
  - Contains arbitrary data stored in zero or more data blocks
  - Treated as stream of bytes by the system

- **Directory**
  - Contains list of file names along with pointers to associated nodes (index nodes, or *inodes*)
  - Organized in hierarchies

- **Special**
  - Contains no data, but serves as a mapping to physical devices
  - Each I/O device is associated with a special file

- **Named pipe**
  - Implement inter-process communication facility in file system name space

- **Link**
  - Provides name aliasing mechanism for files

- **Symbolic link**
  - Data file containing the name of file it is linked to
Navigating the File System

- Show contents of present directory:
  
  % ls -l

- Print absolute path to current directory:
  
  % pwd

- Move into a directory:
  
  % cd dirname

- Move one directory up:
  
  % cd ..

- Move directly to your home directory:
  
  % cd ~
Copying and Moving Files

• Copy fileA to fileB in same directory:

  % cp fileA fileB

• Copy fileA in directory above to current directory:

  % cp ../fileA fileB

• Copy entire directoryA as a subdirectory to directoryB

  % cp -r directoryA directoryB/

• Rename fileA to fileB

  % mv fileA fileB

• Rename directoryA to directoryB

  % mv directoryA directoryB
Do One Thing Well

- This is the first rule from the *UNIX Philosophy* on utilities.
- It means that a bunch of small, specialized utilities provide for more flexibility and power than large, multifunction programs.
- Examples,
  - `cd`, change directory
  - `cat`, dump contents of a file
  - `find`, find a file based on some criteria
  - `file`, tries to identify a file type
  - `grep`, finds a text pattern in a stream of text
  - `head`, output the first N lines of a file
  - `tail`, output the last N lines of a file
  - `less`, allows one to page through output of another command
  - `touch`, creates an empty file
  - `wc`, counts characters, words, and lines in a text file (or input stream)
  - ... there are many, many more very useful specialized utilities..
Everything Is a Filter

• This is the second rule from the *UNIX Philosophy* on utilities.
• It means that a utility should accept input via one channel (STDIN) and output via another channel (STDOUT).
• This allows for many specialized commands to be linked together in some very interesting and useful ways.

• Examples 1, simple redirection into a *cat*:
  – % cat myfile.txt | grep "hello"
  – % grep "hello" < myfile.txt

• Example 2, redirecting output into a file:
  – % cat myfile.txt | grep "hello" > newfile.txt
  – % grep "hello" < myfile.txt > newfile.txt

• Example 3, many utilities inlined:
  – % cat myfile.txt | grep "hello" | grep "world" | less
Shell Programming

• a.k.a., “batch” programming because it is scripted, not real time interaction with the shell.
• The ability to shell script will provide one with a distinct advantage when working on *nix systems.
• Shell scripts fill a very real void that even scripting languages (Perl, Python, etc) do not cover, so there is a place for them even today.
• Shell scripts are very useful for automating tedious system administration tasks, but can be more general purpose.
• In general, a shell script should be used to link existing executables and tools together...tools that do one thing well (sound familiar?).
When Not to Shell Script

- Shell scripts are very useful and can be the perfect solution for many problems, as long as they are the right kind of problems.
- One should not use a shell script to write a program when:
  - most of the functionality is programmed internally, or
  - what you wish to do is complex and/or requires a lot of math or string manipulations
  - what you need to do can't be expressed using loops and combinations of existing (or custom) special purpose utilities
• The most common utility is called, \texttt{vi}.
• Everyone should know how to use \texttt{vi}, and this will be covered more carefully in the lab.
• \texttt{vi} is completely keyboard driven (no mouse!).
• \texttt{vi} operates under specific \textit{modes}; for example, there is \textit{insert} mode and there is \textit{read} mode.
• When in read mode, one literally navigates the cursor wrt the text.
• Consider the text file a 2d grid, where the cursor may be moved up, down, left, and right.
Editing A Text File

- `vi` must be put in “insert” mode in order to write anything.
- When in doubt about what the current mode is, hit `<esc>` and start over.
- There are 2 ways to do this:
  - `<esc>` `i` (insert before current char)
  - `<esc>` `a` (insert after current char)
  - `<esc>` `o` (create new line below, be in insert mode)
  - `<esc>` `O` (create new line above, be in insert mode)
- Save, quit:
  - `<esc>` `:w` # save
  - `<esc>` `:wq` # save, then quit
  - `<esc>` `:q!` # quit, don't save
- `vi` provides for many other typical capabilities, such as:
  - cut/delete a line (or N lines)
    - `dNd` (N is some number > 0)
    - `dd == d1d`
  - copy
    - `yNy` (N is some number > 0)
    - `yy == yly`
  - paste
    - `p` (paste below current line)
    - `p` (paste above current line)
• *UNIX is user friendly, it's just picky about its friends.* (anon)
• Be persistent (force yourself to use UNIX), and you will be friends with UNIX.
• The shell is the main venue of user interaction, so be sure to understand what the user *environment* is.
• UNIX utilities are small and special purposed; UNIX provides the means to string these commands together in very powerful ways.
Summary pt. 2

• The file system on UNIX is presented as a unified tree with a single root. There are still directories and files, just no “C” drives, etc.

• There are easier editors for beginners, but none as powerful as \textit{vi}. Learn to use \textit{vi}, and try to do all of your programming assignments using it.

• Lastly, it would do you well to revive an old machine with some “\textit{freeNIX}” (linux, freebsd, etc) to use daily – the benefits of this can not be over emphasized.
Credits

• B. Estrade
• Chirag Dekate

Addition Resources

• http://www.freebsd.org (a free BSD based *nix)
• http://www.ubuntu.org (a free, popular Linux distribution)